移动端 Touch 事件梳理和踩坑

前言

在移动端开发中，几乎无法避免 Touch 事件，然而每次遇到 Touch 事件，总是能把人搞头大，在工位上 **猿鸣三声泪沾裳**🙈，为了避免悲剧继续传播，今天一起来梳理一下 Touch 事件的相关内容和踩坑总结吧！

Touch 事件基础

TouchEvent

- `touchstart`：当手指触摸屏幕的时候触发，即使已经有一个手指放在屏幕上也会触发
- `touchmove`：当手指在屏幕上滑动的时候连续地触发
- `touchend`：当手指从屏幕上离开的时候触发
- `touchcancel`：当一个或多个触摸点以特定于实现的方式被中断（例如，创建了太多触摸点）

TouchList

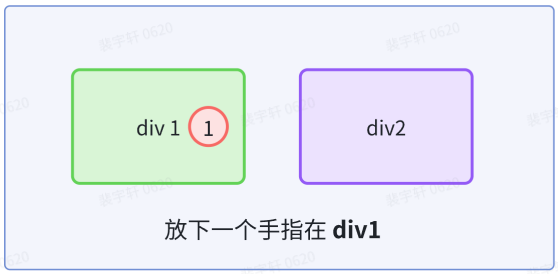
- `touches`：当前屏幕上所有触摸点的集合列表
- `targetTouches`：绑定事件的元素上的触摸点的集合列表
  - 这个属性通常用来判断是否有手指触摸了某个元素，并且可以用来做一些与之相关的交互
- `changedTouches`：触发事件时改变的触摸点的集合
  - 对于 touchstart 事件，列出在此次事件中新增加的触点
  - 对于 touchmove 事件，列出和上一次事件相比较，发生了变化的触点
  - 对于 touchend ，列出离开触摸平面的触点

```
// touchend 触发时
e.touches[0] // undefined
e.changedTouches[0] // TouchEvent
```

光看解释没看懂，三者啥区别呢？

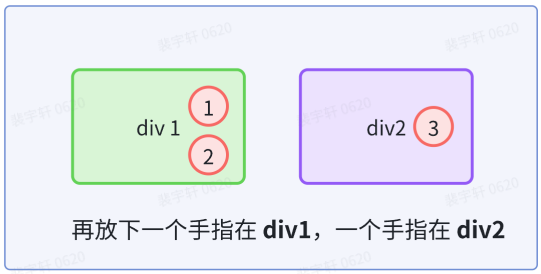
表示触点

第一次



- touches: [1]
- changedTouches: [1]
- targetTouches:
  - 对于 **div1**，targetTouches 为 [1]
  - 对于 **div2**，targetTouches 为 []

第二次



- touches: [1, 2, 3]
- changedTouches: [2, 3]
- targetTouches:
  - 对于 **div1**，targetTouches 为 [1, 2]
  - 对于 **div2**，targetTouches 为 [3]

手势滑动原理

在移动端中，我们可以通过监听 touchstart / touchmove / touchend 事件，获取手势偏移量，配合 CSS transfrom 动态调整 `transform: translate` 的位移值，从而实现一些动画效果，比如下面的一个例子：

获取当前坐标：

```
const onTouchStart = (e: TouchEvent) => {
    console.log(e.touches[0].pageX);
    console.log(e.touches[0].pageY);
}
```

获取偏移量：

```
const getX = e => e.touches[0].pageX

const onTouchStart = () => {
    const startX = getX(e);
}

```

```
const onTouchMove = (e: TouchEvent) => {
    const curX = getX(e);

    // 当前偏移量
    const offsetX = curX - startX;
}
```

设置偏移量：

```
<div
    className="touch-wrapper"
    style={{
        // 0.5 是一个阻力值，比如滑动 100px，实际 tab 只移动 50px
        transform: `translate(${offsetX * 0.5}px, 0)`,
    }}
>
    ...
</div>
```

原理很简单，但是在实际的开发中，却经常遇到 **意料之外** 的问题，下面我们来简单梳理一下~

都有那些坑?

坑1：动画卡顿

1. JS 执行引起渲染阻塞

在使用一些框架如 React 来进行开发的时候，我们常常会将获取到的偏移量 offsetX，通过 state 来保存：

```
const [offsetX, setOffsetX] = useState();

const onTouchMove = (e: TouchEvent) => {
    // 当前偏移量
    const offsetX = curX - startX;

    setOffsetX(offsetX);
}
```

然后再通过数据再触发视图更新：

```
<div
    style={{
        transform: `translate(${offsetX}px, 0)`,
    }}
>
    ...
</div>
```

问题：当挂载 touch 事件的元素 el 中 dom 结构变复杂时，页面可能会发生肉眼可见的卡顿，原因是每次 setState 都会触发一次视图渲染，执行额外的 JS， **JS 执行有可能导致了渲染阻塞**，因此发生了卡顿

解决办法：直接设置元素的 style，避免执行额外的 JS

```
const setPosition =
    (offset: number) => {
```

```
        if (wrapperRef.current) {
            wrapperRef.current.style.transform = `translate(${-offset}px, 0)`;
        }
    }
```

2. touchmove 时设置 transition

问题：有时候，在 touchend 事件触发后，我们希望展示一个过渡动画，比如轮播图平滑切换到下一页，对于这种情况，我们一般使用 transition 来实现。然而，当 touchmove 触发时，由于元素偏移量会频繁变动，而每次变动在浏览器帧率刷新时都会触发 0.3 秒的 transition过渡动画，因此会看到元素频繁地发生抖动

```
<div
    className="touch-wrapper"
    style={{
        transition: 'transform .3s',
    }}
>
    ...
</div>
```

解决办法：在 touchstart 时，设置 transition 为 none；在 touchend 时再设置 transition 属性

```
<div
    className="touch-wrapper"
    style={{
        transition: touching ? 'none' : 'transform .3s',
    }}
>
    ...
</div>
```

坑2：浏览器默认事件触发

问题：当 touchmove 事件触发的时候，在 webview 或者 手机浏览器内，可能会触发一些默认的事件，如：

- 手势返回操作
- 浏览器下拉刷新
- 浏览器回弹效果
- 默认滚动

解决办法：

方法一：preventDefault

```
const move = (e: TouchEvent) => {
    e.cancelable && e.preventDefault();
};


// 必须使用 addEventListener 来完成监听，不可以挂载到 JSX 元素上
el?.addEventListener('touchmove', move, { passive: false });
```

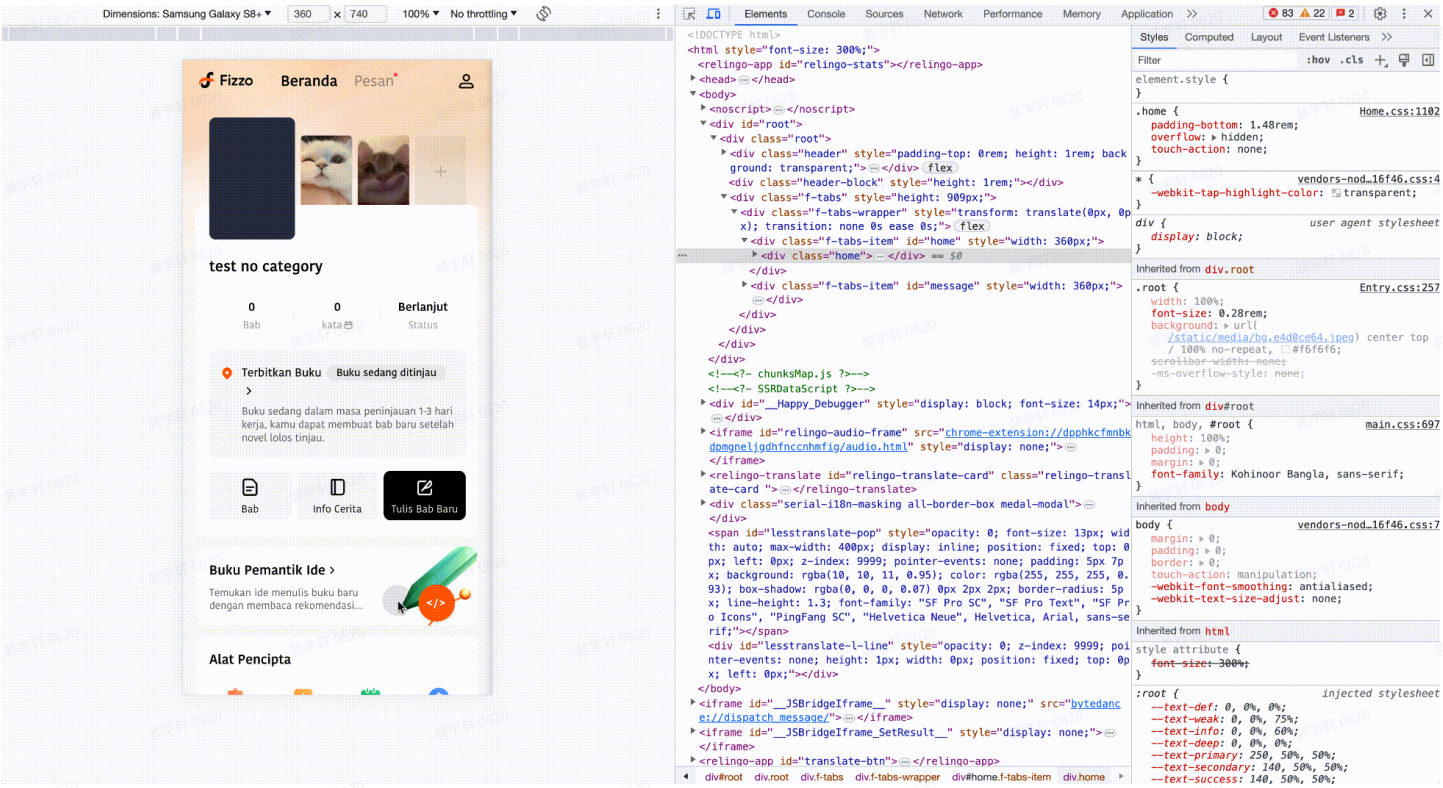
- **e.cancelable 为 true**，调用 `preventDefault()` 能生效
- **e.cancelable 为 false** ，意味着不存在默认动作或无法阻止该元素的默认动作，如果强行调用 `preventDefault()` 的话会触发以下的报错，并且默认事件也无法被取消：

```
▸ 10 [Intervention] Ignored attempt to cancel a touchmove event with cancelable=false, for example
     because scrolling is in progress and cannot be interrupted.
```

方法二：touch-action

如果 `preventDefault()` 无法成功阻止默认事件时，可以尝试设置 `touch-action` 来阻止浏览器的默认事件。`touch-action` 用于指定某个区域内是否允许用户操作，以及如何响应用户操作，该属性用于取消浏览器默认手势行为，开发人员自定义滚动和手势行为

在下图中，设置 `touch-action: none` ，页面将无法滚动：

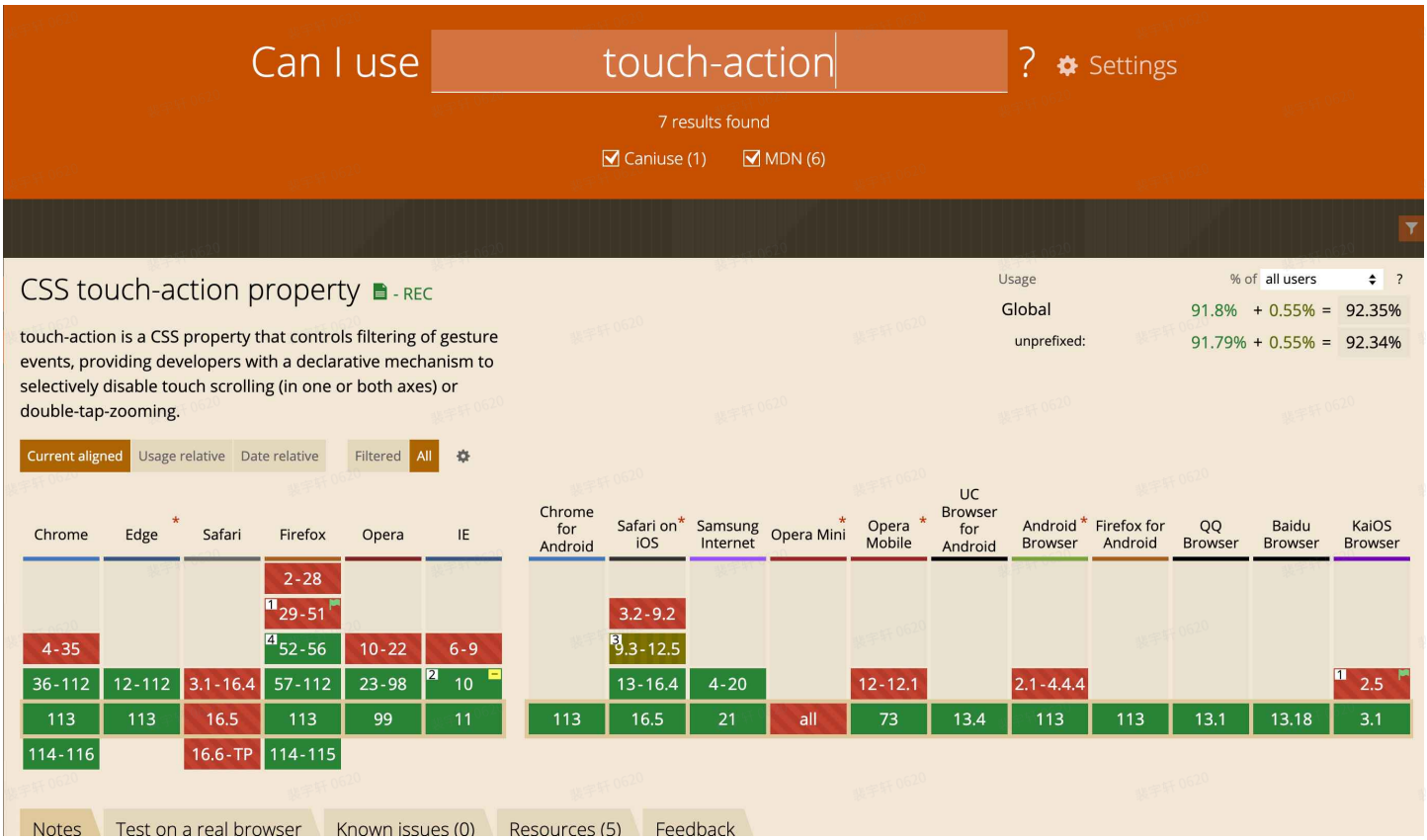


MDN：https://developer.mozilla.org/en-US/docs/Web/CSS/touch-action

```
touch-action: pan-y; // 只启用 y 轴手势，禁用 x 轴手势

auto ： 当触控事件发送在元素上时，由浏览器来决定进行那些操作，比如viewport进行平滑 缩放。
none ： 当触控事件发生在元素上时，不进行任何操作
pan-x ： 启用单指水平平移手势
pan-y ： 启用单指垂直平移手势。
manipulation ： 只可以进行滚动和持续缩放操作。如双击缩放等别的手势
pinch-zoom ： 启用多手指平移和缩放页面，这可以和任何平移值组合
```

兼容性如下："touch-action" | Can I use... Support tables for HTML5, CSS3, etc



注意：不管调用 `preventDefault()`、还是使用 `touch-action`，都意味着你需要阻止浏览器的默认行为，这可能会导致一些意外情况，比如页面内文本域无法异常

Q & A:

😉 **为什么 e.cancelable 为 false?**

1. **监听器的 passive 为 true**

根据 MDN 的描述，passive 为 `true` 意味着不能调用 `preventDefault()` 来阻止被绑定事件元素的默认行为；在一些特殊的浏览器中 passive 默认为 `true`，如 Safari，因此 `preventDefault()` 可能无法被调用

解决办法：将 passive 设置为 false

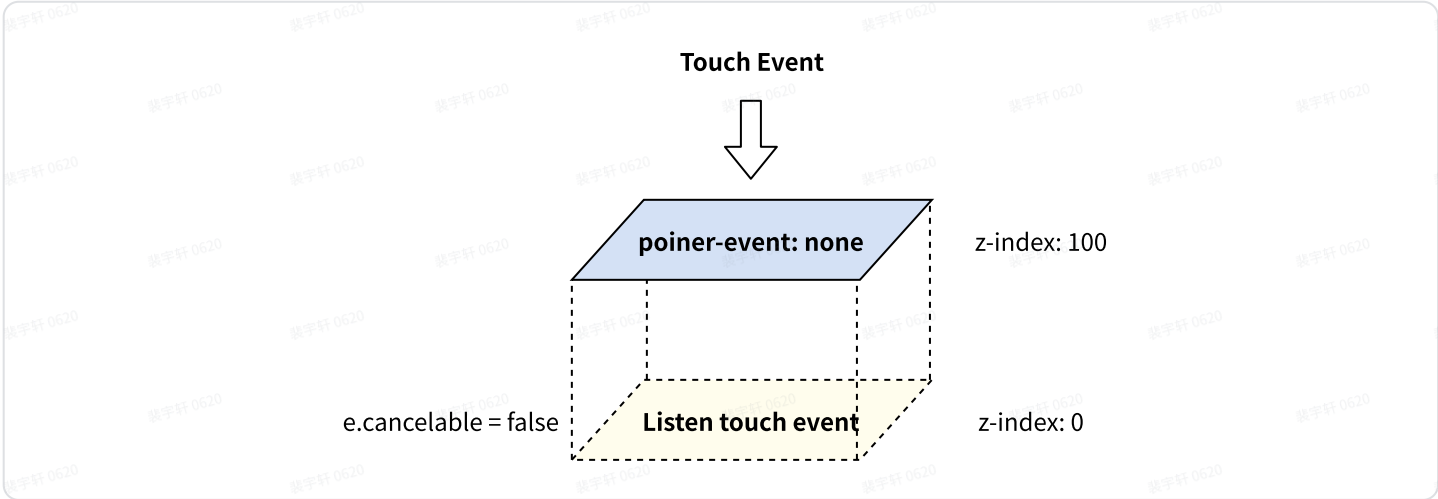
```
dom.addEventListener('touchstart', start, { passive: false });
dom.addEventListener('touchmove', move, { passive: false });
dom.addEventListener('touchend', end, { passive: false });
```

2. **高 z-index 元素覆盖了事件，但是设置了 pointer-events: none**

在下面这个例子中，遮罩层为 `div.mask` ， `div#touch` 则监听了 `touchmove` 事件

- 正常情况下， `div.mask` 会中断事件向 `div#touch` 传递
- 但是 `div.mask` 同时设置了 `pointer-events: none` ，这让 Touch 事件可以传递到 `div#touch`
- 但是在部分场景下，如 chrome devtool 中，虽然 `div#touch` 能够监听 Touch 事件触发，但是 `e.cancelable` 却为 `false`

当然，这种 case 还是比较少见，在大多数浏览器中是正常的




```
<body>
    <div id="root">
        <div id="touch"></div>
    </div>
    // z-index: 1000
    // pointer-events: none
    <div class="mask"></div>
</body>
```

解决办法：设置 touch-action

😉 preventDefault “不生效”？

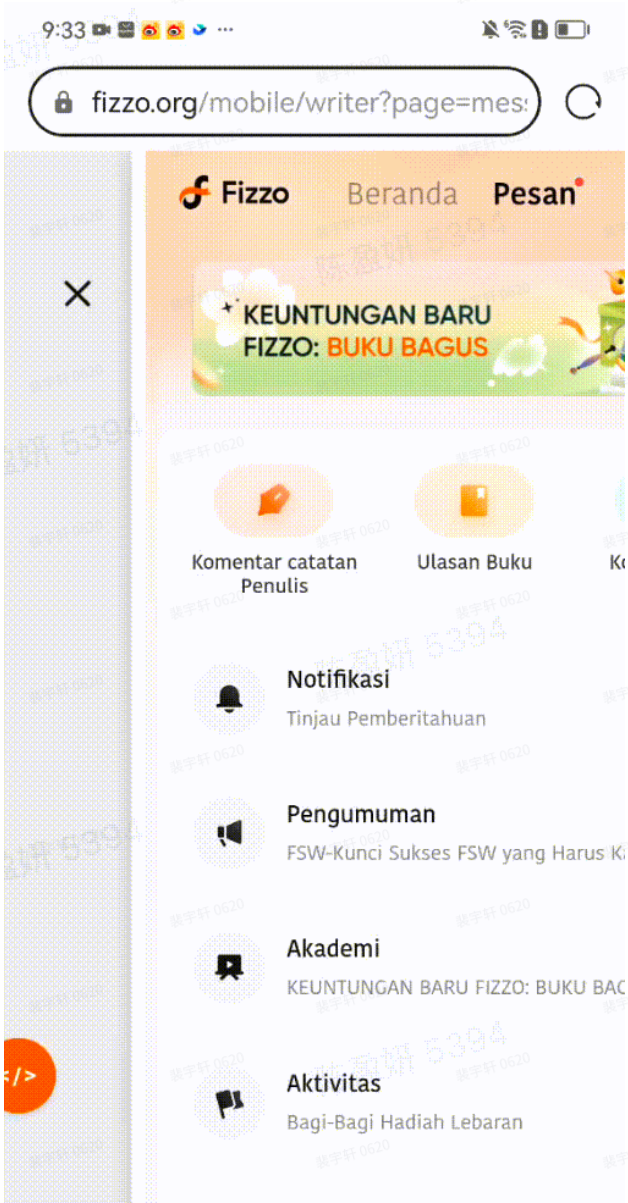
1. `preventDefault()` 不连续

有时候，遇到事件频繁触发的场景，我们可能会想着去给事件 handler 添加上一个 throttle 函数，以适应帧率的变化。然而当给 touchmove 事件添加了一个 throttle 函数时，再调用 `preventDefault()` ，可能还是会让浏览器的默认行为发生，因为 `preventDefault()` 的调用是不连续的

2. `overflow: scroll` 导致 `preventDefault()` 失败

可能存在一种情况：在部分安卓手机中，x方向滑动触发时，`preventDefault()` 无法取消 y方向的滑动

原因：有可能滑动容器设置了 `overflow: scroll`

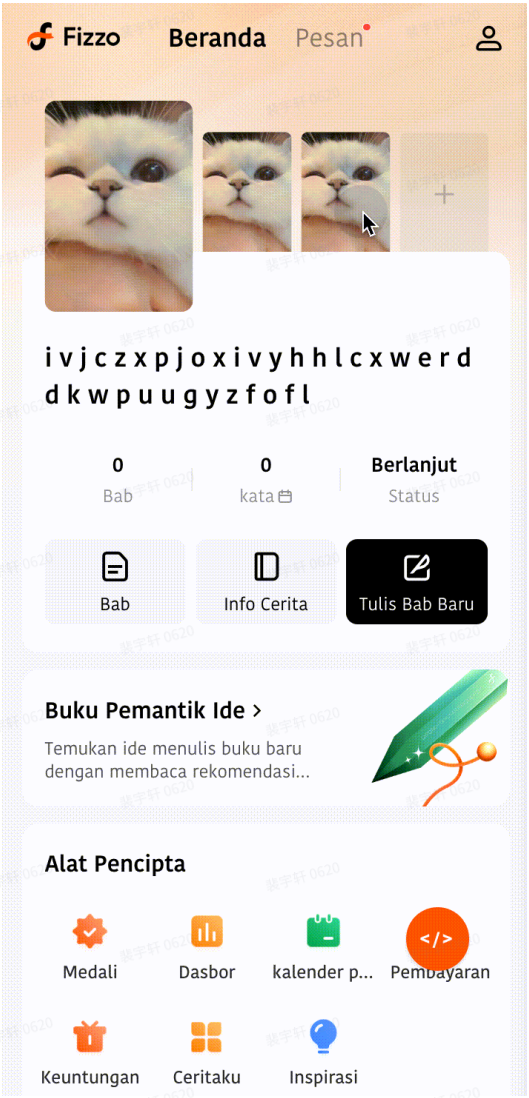


解决方法：

1. 去除 `overflow: scroll`
2. 或者加上 `touch-action` 属性

坑3：父子元素 Touch 区域重合

问题：在父子元素同时挂载 Touch 事件的时候，会同时执行两个 Touch 事件，比如下图中。在这种情况下，我们希望子组件触发 Touch 事件时，父组件的 Touch 事件不触发



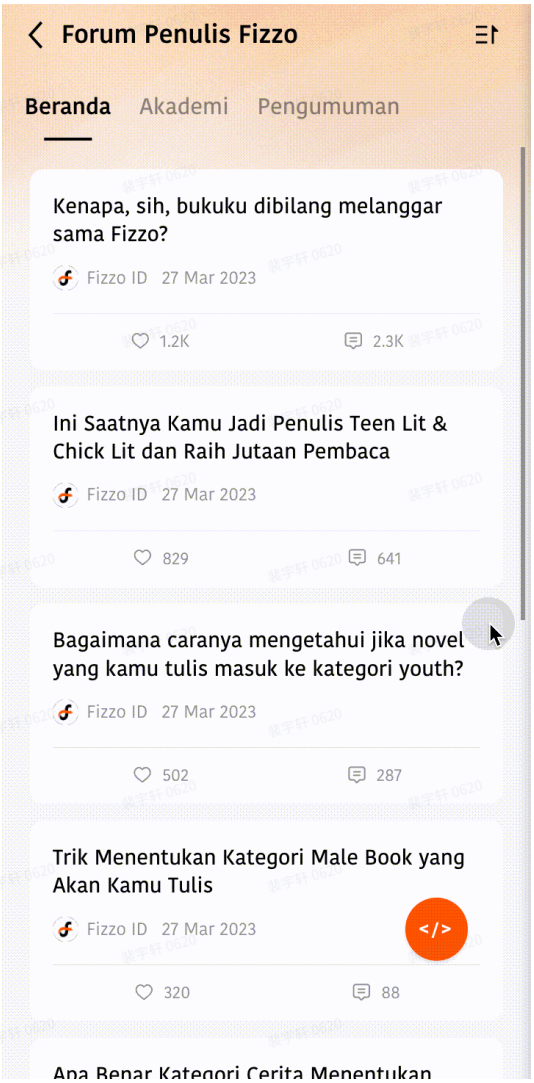
解决办法：在 touchstart 和 touchmove 时使用 `stopPropagation()` 来防止事件冒泡

```
const start = (e: TouchEvent) => {
    e.stopPropagation();
};

const move = (e: TouchEvent) => {
    e.stopPropagation();
};
```

坑4：不同组件多个方向滑动控制

问题：当一个页面同时存在不同组件多个方向的滑动时，如同时存在 PullRefresh（下拉刷新）组件和 Tab 组件时，可以会遇到这样的问题：**当手势向左下方滑动时，PullRefresh 会向下滑，同时 Tab 会向左滑**



解决办法：这个时候，我们需要判断手势滑动的方向，当确定手势滑动方向为 `X` 轴 的时候，才设置 Tab 组件的位移；当确定手势滑动方向为 `Y` 轴 的时候，才设置 PullRefresh 组件的位移😭

```
enum Direction {
    Up,
    Down,
    Left,
    Right
}

const onTouchMove = (e: TouchEvent) => {
    const { startX, startY, direction } = touchInfoRef.current;

    const curX = getX(e);
    const curY = getY(e);

    const offsetX = Math.floor(curX - startX);
    const offsetY = Math.floor(curY - startY);

    // 第一次 touchmove 触发，设置位移的 direction
    if (direction === undefined) {
```

```
        const dy = Math.abs(offsetY);
        const dx = Math.abs(offsetX);

        // dx > dy，说明 x 轴的位移大于 y 轴
        if (dx > dy && dx > 0) {
            touchInfoRef.current.direction = offsetX > 0 ? Direction.Left : Direction.Right;
        }
        // dx < dy，说明 x 轴的位移小于 y 轴
        if (dx < dy && dy > 0) {
            touchInfoRef.current.direction = offsetY > 0 ? Direction.Down : Direction.Up;
        }
    }

    console.log(touchInfoRef.current.direction);
};
```

- `touchmove` 触发，**未设置 `Direction`** ，设置 `Direction`
  - `Y` 轴位移 > `X` 轴位移，说明滑动方向在 `Y` 轴
    - 如果 `offsetY` > 0，`Direction` 为 `Left`
    - 如果 `offsetY` < 0，`Direction` 为 `Right`
  - `Y` 轴位移 < `X` 轴位移，说明滑动方向在 `X` 轴
    - 如果 `offsetX` > 0，`Direction` 为 `Down`
    - 如果 `offsetX` < 0，`Direction` 为 `Up`
- `touchmove` 触发，**已设置 `Direction`** ，复用之前设置的 `Direction`

🤔 第三方库的 touch 事件如何控制?

有时候，PullRefresh 、Tab 可能是一个第三方库的组件，Touch 相关的逻辑在组件内部实现，虽然可以从外部判断滑动方向，但是即使判断了滑动方向，也无法对组件内部的 Touch 事件来进行控制：

如何解决?

理想情况 提 pr 修复或者 fork 代码来修改

一般来说像横滑的时候也会触发下拉这种情况，其实是组件设计的时候有所欠缺，最好的方式是提 pr 或者 fork 一份，给组件加上方向控制

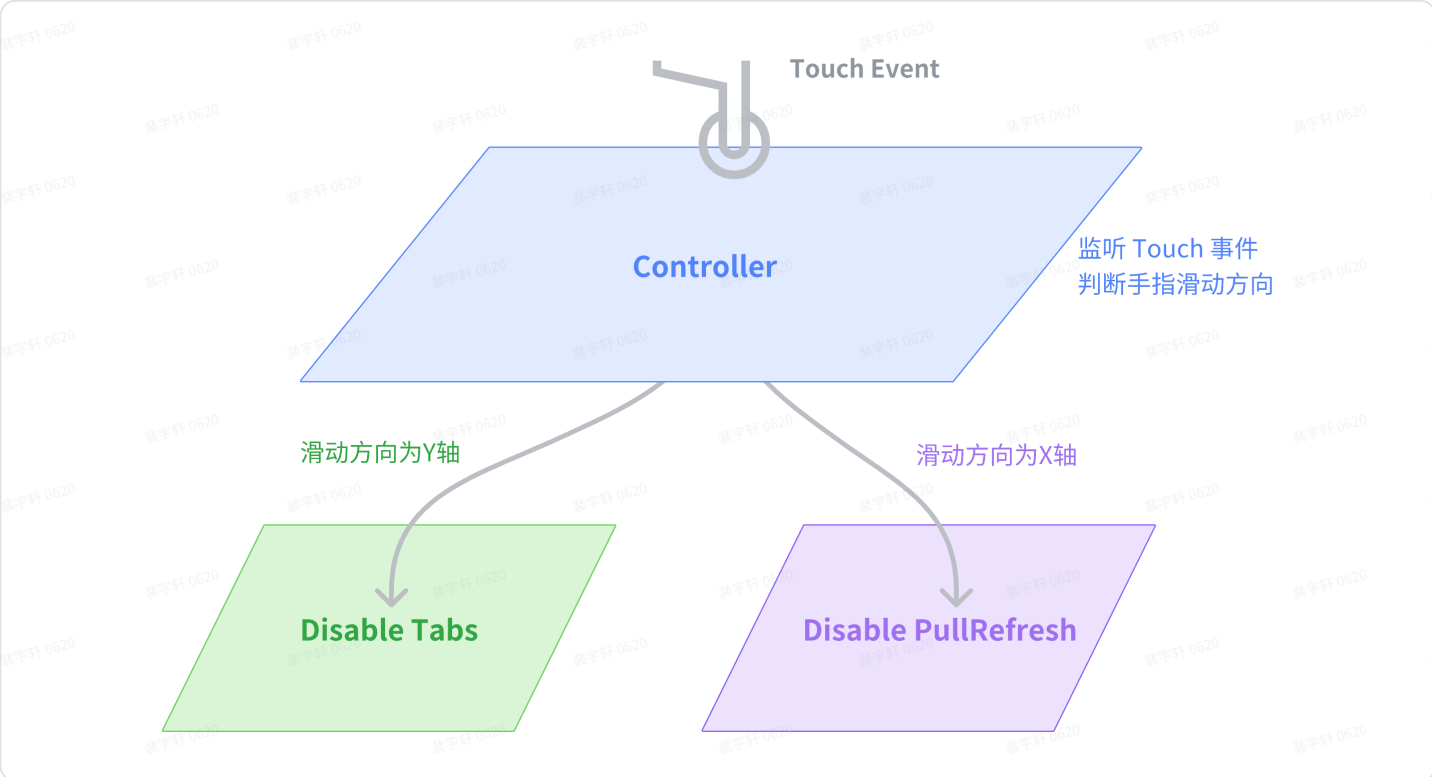
比较推荐 通过组件参数控制

- **方法一**：通过组件提供的 `onTouchStart` 、 `onTouchMove` 等钩子控制滑动方向

如果组件提供 `onTouchStart` 、 `onTouchMove` 等钩子，一般这种钩子都会要求使用的时候返回一个 boolean，从而去判断这个这个事件是否继续执行，我们在这些钩子中判断手指滑动的方向，然后通过钩子的返回值来禁止某个方向的滑动

- **方法二**：通过组件类似 `disableTouch` 的属性控制

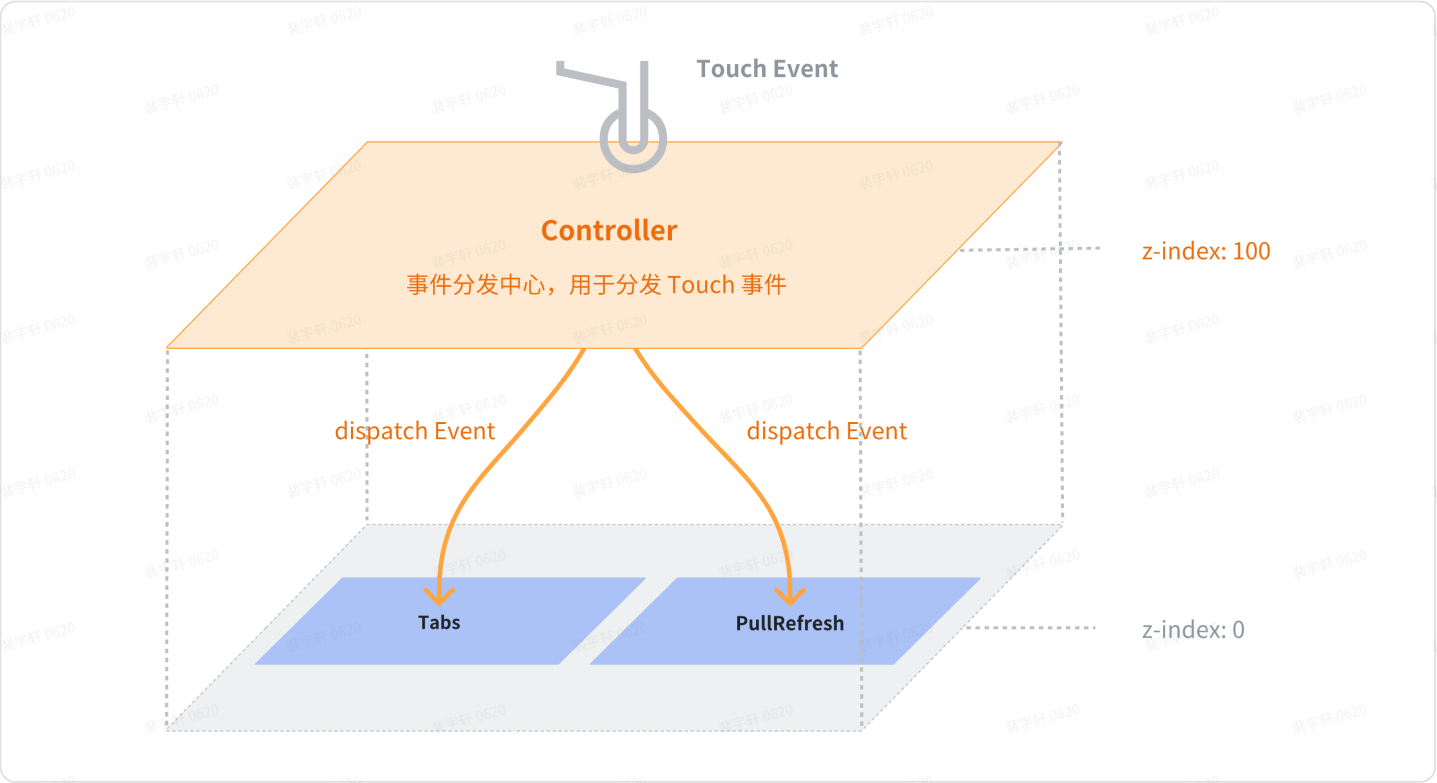
如果组件有 `disableTouch` 类似的参数，可以在业务中再设置一个 touch 的控制层，在这个控制层中，可以判断touch 的方向，然后 disable 掉对应组件的事件。比如产生 x 轴滑动时，禁用 PullRefresh 的 touch 事件；产生 y 轴位移的时候，禁用 Tab 的 touch 事件。



不推荐：通过 `dispatchEvent` 来进行事件的分发

思路：覆盖 PullRefresh 、Tab 原来的 Touch 事件，通过 `dispatchEvent` 来分发事件

缺点：所有事件都被 mask 屏蔽了，需要将 scroll、Touch 等事件重新分发，并且可能会遗漏掉某些事件，比较 trick



1. 通过 `z-index` 来创建一个 蒙层，从而覆盖 PullRefresh 、Tab 的事件，这个蒙层专门用于监听 Touch 事件

```
<div class="touch-action"></div>

.touch-action {
    position: absolute;
    top: 0;
    bottom: 0;
    left: 0;
    right: 0;
    z-index: 100000;
}
```

2. 在蒙层中监听 Touch 事件，判断滑动方向，并且将 Touch 事件分发给 PullRefresh 和 Tab

```
const el = document.querySelector('.touch-mask');
el?.addEventListener('touchmove', e => {
    ...
    if (aixs === 'x') {
        // 分发给对应的元素
        const dom = document.querySelector('.tabs');
        const ev = new TouchEvent(e.type, e);
        dom?.dispatchEvent(ev);
    }
```

```
10 });
```

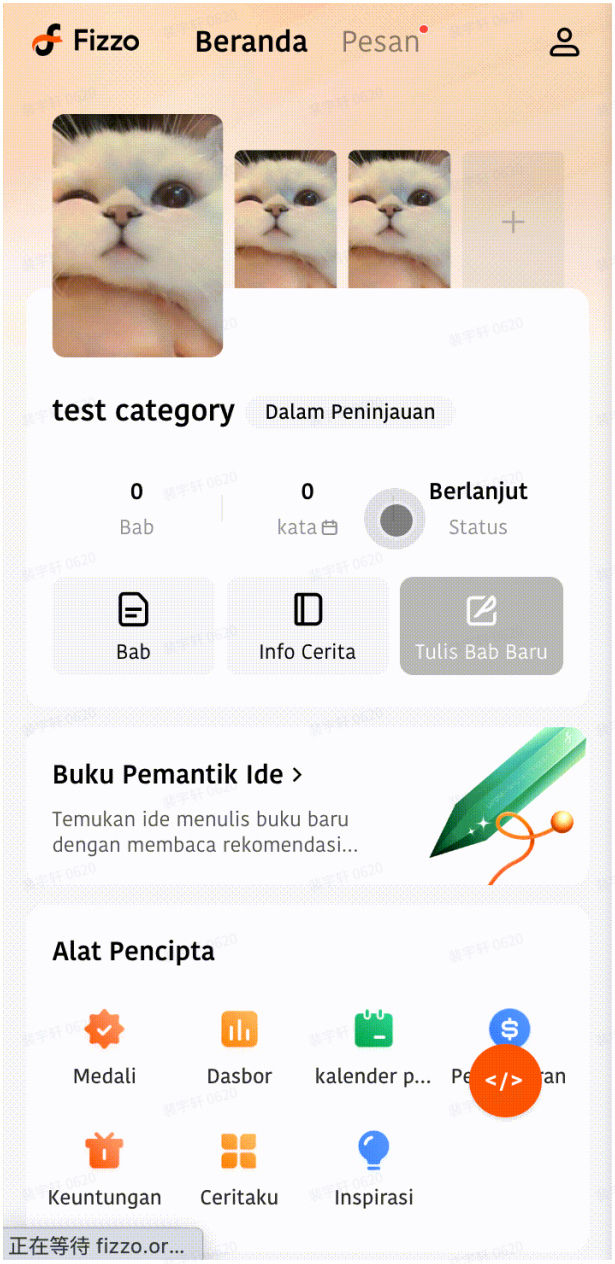
坑5：Touch Event 不触发

1. `e.target` 被移除

问题： `e.target` 在 touchstart 或 touchmove 的时候被移除的时候不会触发 touchend 。如下图例子，子组件在 loading 的时候，父组件触发了 touchmove，此时子组件 loading 完成，`e.target` 被移除，此时父组件的 touchend 不会被触发

> Note that if the target element is removed from the document, events will still be targeted at it, and hence won't necessarily bubble up to the window or document anymore. If there is any risk of an element being removed while it is being touched, the best practice is to attach the touch listeners directly to the target.

详细解释：Touch: target property - Web APIs | MDN



解决方法：将监听器直接绑定到 `e.target`

> stackoverflow上找到的相关答案：Touch Move event don't fire after Touch Start target is removed

```
el.addEventListener("touchstart", e => {
    const onTouchMove = () => {
        // handle touchmove here
    }
    const onTouchEnd = () => {
        e.target.removeEventListener("touchmove", onTouchMove);
        e.target.removeEventListener("touchend", onTouchEnd);
        // handle touchend here
    }
    e.target.addEventListener("touchmove", onTouchMove);
    e.target.addEventListener("touchend", onTouchEnd);
```

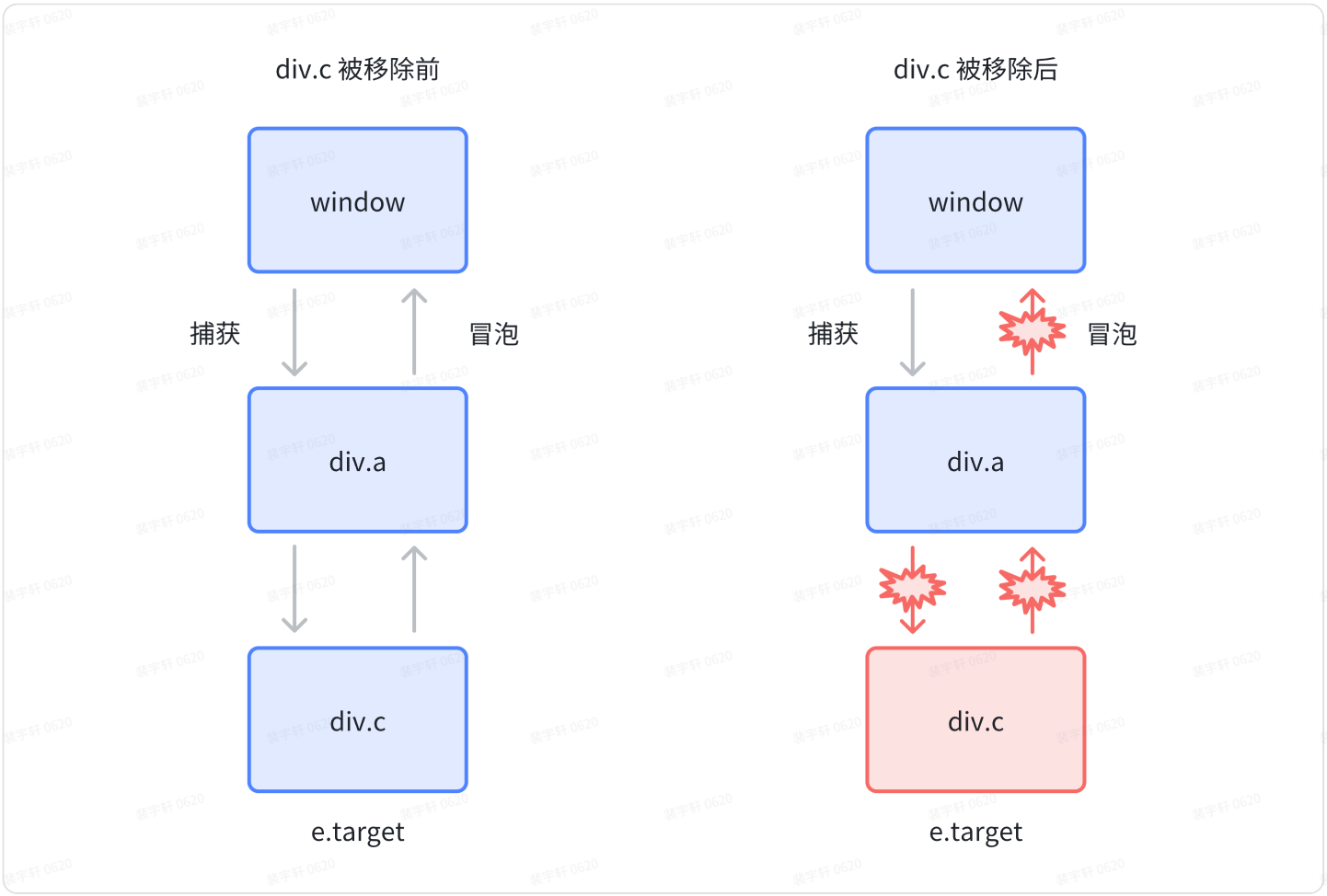
```
    // handle touchstart here
});
```

注意：这种方式可能导致 `e.stopPropagation()` 失败，如下，如果需要防止事件冒泡到父元素，可以让子元素在 `touchstart` 的时候调用 `e.stopPropagation()`

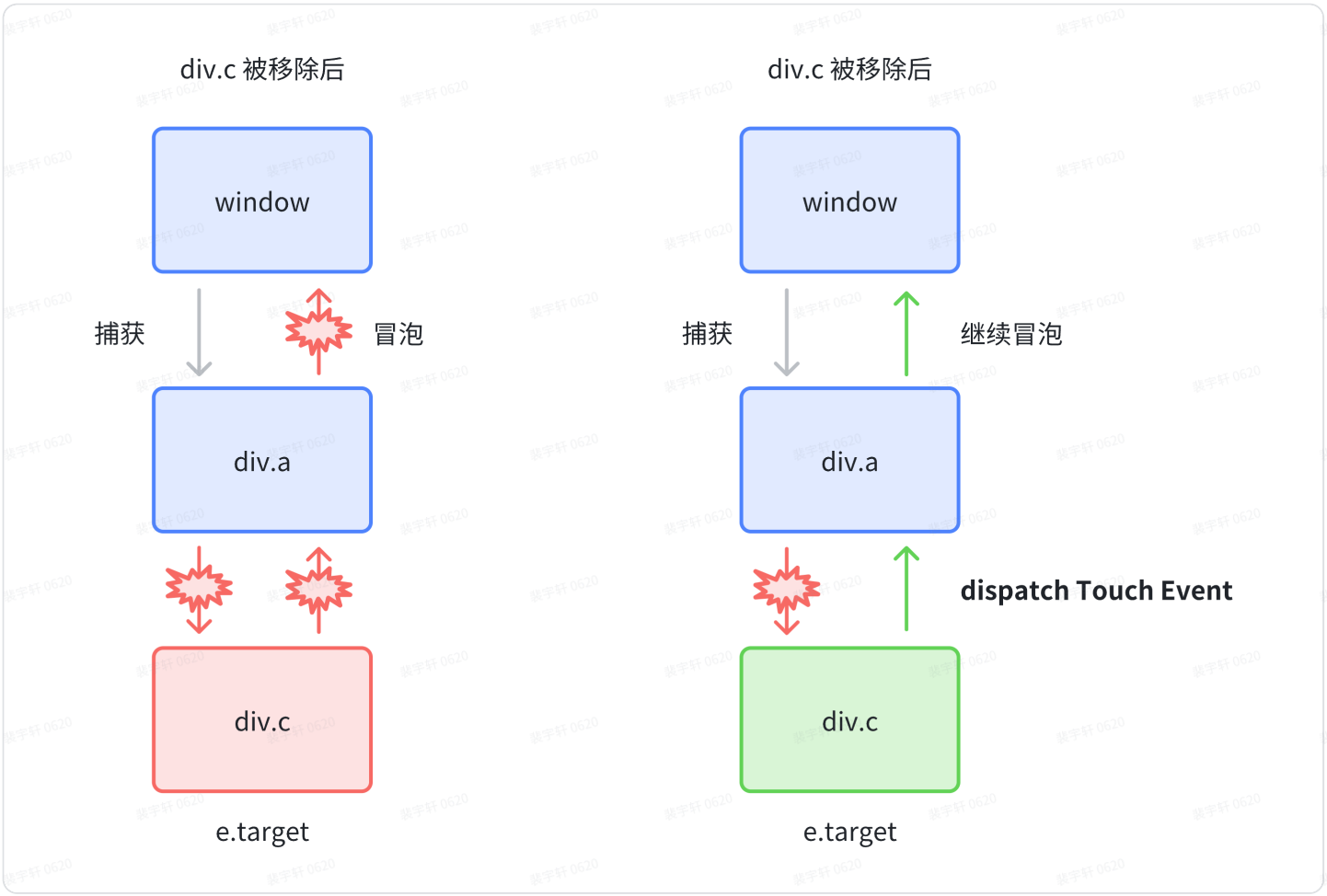
```
// 子元素
childEl.addEventListener("touchmove", e => {
    e.stopPropagation();
});

// 父元素
fatherEl.addEventListener("touchstart", e => {
    e.target.addEventListener("touchmove", e => {
        // 因为：父元素 e.target === 子元素 e.target
        // 所以：即使子元素调用了 stopPropagation，“father touchmove”还是会被 log 出来
        console.log('father touchmove')
    });
});
```

补充：像上面这样其实还是有问题，虽然 Touch 事件继续在 `e.target` 上触发 了，但是事件却不会继续冒泡了

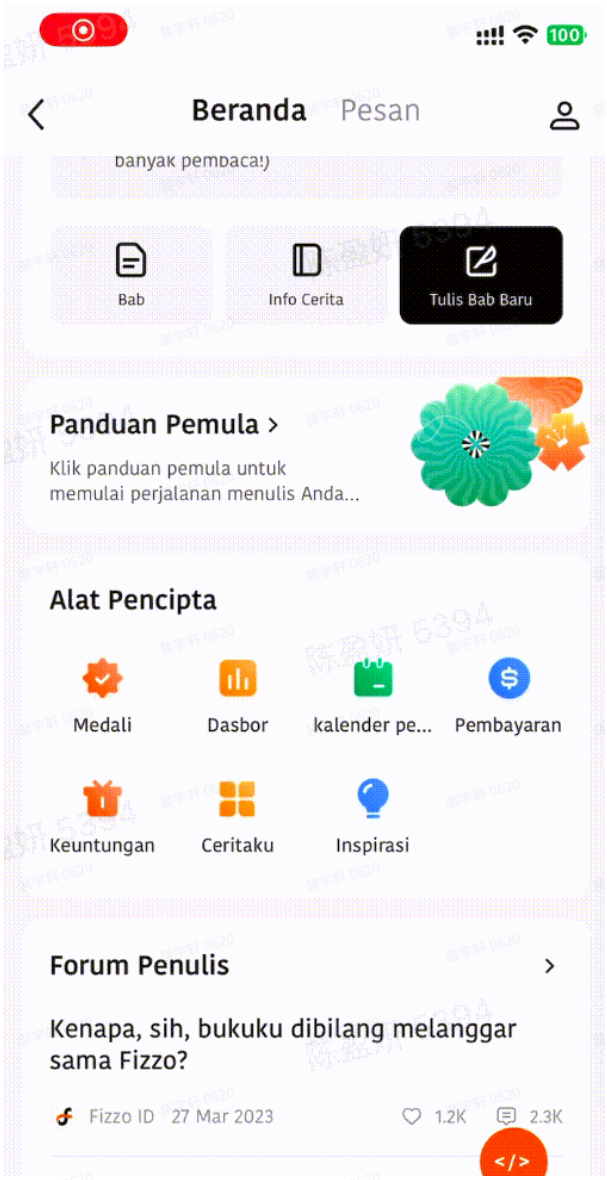


如果需要继续将 Touch 事件冒泡到上一层，可以在 `e.target` 的 Touch 事件中，通过 `dispatchEvent` 将事件转发至当前存在的父级元素中，使事件继续冒泡



2. 部分浏览器特定行为不执行 touchend

问题：在进行一些特定行为时，部分浏览器可能不会触发 `touchend`，但是会触发 `touchcancel`，比如 iOS webview 内上滑切换应用程序的时候，便不会触发 `touchend`



解决办法：用 `touchcancel` 代替 `touchend`

3. 未调用 `preventDefault()`

问题： `touchstart` 或者第一次 `touchmove` 未调用 `preventDefault()`，可能会导致 `touchend` 不触发

> To workaround this bug you have to call preventDefault() on either the touchstart or first touchmove event. Of course, this prevents the native scrolling, **so you will need to re-implement that yourself.**、

解决办法：在 touchstart 或者第一次 touchmove 调用 `preventDefault()`。这是一个比较久远的浏览器 bug 了，在 Android v4.1 中已得到修复，基本不用关注

最佳实践

> Using Touch Events - Web APIs | MDN

当你使用 touch 事件的时候，这里有一些最佳实践：

- 最小化在 touch 事件中执行工作
- 将 touch 事件绑定到特定目标元素（而不是整个文档或文档树中较高的节点）
- 在 `touchstart` 中绑定 `touchmove`、 `touchend`、 `touchcancel`
- 目标触摸元素或节点应足够大，以适应手指触摸。如果目标区域太小，则触摸时可能会导致相邻元素触发其他事件

封装一个 useTouch

功能 list

- [Feat] **上下左右方向** 判断
- [Feat] **滑动坐标轴** 判断
- [Feat] **禁止事件冒泡**
- [Feat] **指定方向阻止默认事件**

兼容处理

- [Fix] 元素被移除后 touch 事件不触发
- [Fix] iOS上滑 `touchend` 不触发

CodeSandBox：useTouch - CodeSandbox